Industrial & Applied Mathematics

4-Vector.org

A Blog with Annotated R-Notebooks: Math, Statistics, and Multiphysics for Learned Peer Discussion on the Science of Data

Optimizing Parameters and/or Hyperparameters in Simulation: A Review of the Susceptible-Exposed-Infectious-Recovered [SEIR] Model of West-Africa Ebolavirus 2013

Michael A. X. Izatt
E | Michael.Izatt@Alum.MIT.Edu (mailto:Michael.Izatt@Alum.MIT.Edu)
E | izatt@UChicago.Edu (mailto:izatt@UChicago.Edu)
W | 4-Vector.org
In | www.linkedin.com/in/max-izatt

May 2, 2020

Version 2020-05-02-0800/MAI

M.A.X. Izatt, “Optimizing Parameters and/or Hyperparameters in Simulation: A Review of the Susceptible-Exposed-Infectious-Recovered [SEIR] Model of West-Africa Ebolavirus 2013”, 4-Vector-org, May 2, 2020, https://4-vector.org/2020/05/04/4vector-org-2020-05-02-0900-mai/ (https://4-vector.org/2020/05/04/4vector-org-2020-05-02-0900-mai/)

Abstract

This paper extends a prior article on applying the Susceptible-Exposed-Infectious-Recovered/Removed (SEIR) mathematical model of infectious diseases to the 2013 West-Africa Ebolavirus epidemic by following Althaus and demonstrates a maximum likelihood / minimum MAPE technique to calculate the model’s coupling constants and transmission rates via simulation. An afterword discusses the applicability of this model to the current COVID-19 pandemic where Immunity of Recovered patients is still being established and is in doubt.

Discussion

This short paper is an extension of a prior illustration of the Susceptible-Exposed-Infectious-Recovered (SEIR) model here:

https://4-vector.org/2020/04/27/4vector-org-2020-04-25-2100-mai/ (https://4-vector.org/2020/04/27/4vector-org-2020-04-25-2100-mai/)

In that writing, we followed Althaus to illustrate the mechanics of fitting the SEIR curve to the data for West-Africa's 2013 Ebolavirus epidemic.

https://4vector.files.wordpress.com/2020/04/althaus_west-africa.pdf (https://4vector.files.wordpress.com/2020/04/althaus_west-africa.pdf)

Althaus published values for the SEIR rates and coupling constants:

- $\frac{1}{\sigma}$: Average Duration of Incubation
- $\frac{1}{\gamma}$: Average Duration of Infectiousness
- β_0: Rate of Transmission in the absence of Public Health Control Intervention Measures
- f: Fatality Rate
- κ: Exponential decay constant for $\beta(t)$, the time-dependent Transmission Rate in the presence of Public Health Control Intervention Measures

---

Analysis

The SEIR model is formulated in terms of the following variables and parameters:

N is the total Population Size

S is the number of Susceptible individuals

E is the number of Exposed individuals

I is the number of Infectious individuals

R is the number of Recovered individuals

C is the total number of Cases

D is the number of Deaths

f is the Fatality Rate.

$\frac{1}{\sigma}$ is the Average Duration of Incubation, so σ is an Interaction Frequency for Exposed subjects to become Infectious

$\frac{1}{\gamma}$ is the Average Duration of Infectiousness, so γ is an Interaction Frequency of Removed persons by Recovery or Death

The total Population $N = S + E + I + R$ where the Population in time $N(t) = N_0 - D$

For modeling purposes, the total Population, N, need not be known exactly if the Population is very much larger than the number of Cases, C. For this illustrative paper that models the outbreak of Ebolavirus in West Africa in 2013/2014, the number of cases was on the order of 1-thousand, so we will set the Population Size to 1-million for simulation purposes.

$$N \gg C$$

The coupled SEIR differential equations are

$$\frac{dS}{dt} = \frac{-\beta(t)\,SI}{N}$$

$$\frac{dE}{dt} = \frac{+\beta(t)\,SI}{N} - \sigma E$$

$$\frac{dI}{dt} = +\sigma E - \gamma I$$

$$\frac{dR}{dt} = (1-f)\,\gamma I$$

The Equations

The time rate of change of Susceptible persons

The first equation says that the time rate of change of Susceptible individuals is negative; that is, in the presence of a contagion, all persons are susceptible, and it is all downhill from there. Individuals cease to be Susceptible once they become Exposed to the contagion, so the time rate of change of Susceptible individuals is monotonically negative. The notation comprises three factors: The Effective Transmission Rate, $\beta(t)$, the current number of Susceptible persons, and a probability of encountering an Infectious person, given by the ratio of the number of Infectious persons to the total Population, $\frac{I}{N}$. The Effective Transmission Rate, $\beta(t)$, encodes the Public-Health Control Intervention Measures such as social distancing, personal protective equipment, hospitalization, vaccination, quarantining, the lot. So the chance of becoming Exposed is controlled by the interaction of the number of Susceptible persons in the population times the Effective Transmission Rate, $\beta(t)$.

However, the migration of a Susceptible individual to Exposed status can only occur upon interaction with an Infectious person, and the probability of encountering such an Infectious person is given by the ratio of Infectious persons to the total Population, $\frac{I}{N(t)}$ This in intuitive; if no individuals are Infectious, there is no possible chance of encountering one, and the time rate of change of Susceptible persons will be zero. Contrariwise, if the entire population is Infectious such that $I = N$, then the time rate of change of Susceptible persons is completely governed by the Public-Health Control Intervention Measures, which are encoded in the $\beta(t)$ expression. If the entire population is Infectious, but you stay in your house say on the International Space Station while wearing your mask, you cannot be Exposed to the contagion. So, there is this intricate dance between the Public-Health Measures and the probability of encountering an Infectious person that governs the time rate of change of Susceptible persons.

The time rate of change of Exposed persons

Individuals migrate from Susceptible to Exposed under the influence of the Effective Transmission Rate and the probability of Infectiousness in the Population, so the first term in the right-hand side of the Exposed time derivative is the additive inverse of the right-hand side of the Susceptible time derivative. Likewise, persons migrate out of the Susceptible compartment/status and into an Infectious state at a frequency σE

The time rate of change of Infectious persons

Individuals migrate from Exposed to Infectious status at the rate σE and likewise from Infectious to ill at the rate γI. The overall compartment of ill persons is of size γI, with two possible outcomes, Death and Recovery; Deaths occur at rate $f\gamma I$ and Recoveries at the rate $(1-f)\gamma I$

Evolution of the Effective Transmission Rate

The Transmission Rate in the absence of Control Intervention Measures is constant: $\beta(t) = \beta_0$

After Control Intervention Measures are introduced at time $t = \tau$,the Transmission Rate at times $t \geq \tau$, is given by the decaying exponential, $\beta(t)$

$$\beta(t) = \beta_0 e^{-\kappa(t-\tau)}$$

Modeling the Epidemic

We assume that the epidemic starts with a single Infected Case, C_0; that is, I_0 = 1 and $C_0 = 1$.

The total number of Infectious Cases, C, and $Deaths$, D, are expressed as

$$\frac{dC}{dt} = \sigma E$$

$$\frac{dD}{dt} = \gamma I$$

The Basic Reproduction Number, R_0, is calculated straight-away as

$$R_0 = \frac{\beta_0}{\gamma}$$

That is, the Basic Reproduction Number is the Initial Transmission Rate in the absence of Control Intervention Measures times the Average Duration of Infectiousness.

The Effective Reproduction Number, R_e is given by

$$R_e = \frac{\beta(t)\, S}{\gamma N}$$

As long as the number of Cases remains much smaller than the number of those who are Susceptible, and as long as Control Intervention Measures work to limit those who are Exposed AND subsequently Infected, then the number of Deaths is small, and most of the entire Population remains Susceptible but not Exposed, then the number of Susceptible individuals constitutes nearly the entire Population, and we can say

$$S \approx N$$

and

$$R_e \approx \frac{\beta(t)}{\gamma}$$

Nelder & Mead have shown that the cumulative numbers of Cases and Deaths are Poisson distributed.

This problem set is solved by numeric integration, termed 'quadrature,' and the solution is the resulting trajectory of the particular variables of interest.

We will be calculating the Effective Transmission Rate, $\beta(t)$, on each iteration, so we will create a function. So as to make the function portable among downstream projects, we will send all required parameters and variables in the methods signature. We could make a call on environment variables and make the method signature less busy, but that would be penny wise and pound foolish; the function would be less intuitive and less portable, so busy as it is, let's send everything that we need when we call the function.

- β_0 is the Initial Transmission Rate
- t is the time, which for the purpose of this illustrative simulation is in units of days
- κ is the exponential decay constant of the Effective Transmission Rate
- τ is the time at which Public-Health Control Intervention Measures are implemented, in units of days

```
# Transmission Rate

# - B_0_ INITIAL TRANSMISSION RATE
# - t_        Current TIME
# - k_        Exponential decay constant due to CONTROL INTERVENTIONS
# - tau_      Time at which CONTROL INTERVENTIONS were instituted by Public-Health officials

FUNC_B_t_ <- function(B_0_, t_, k_, tau_){


  if(t_ > tau_){

    B_0_ * exp(-k_ * (t_ - tau_))

  } else {

    B_0_

  }

}
```

Functions of a Random Variable

For this simulation, we will draw random variates from the applicable Probability Distribution Function (PDF) or Probability Mass Function (PMF) for each continuous or discrete random variable, respectively. There is art and science to making such draws, but the basic rule is do not assume more in the random draw than is in the problem set itself. Here are a few guidelines:

- Empirical/historical data is **always** superior to a theoretical distribution
- In the absence of empirical data, you must become a humble student of statistical distribution theory
- Do not make more claims about the distribution of data than the evidence warrants, and…
- Be very, very cognizant of the properties of a given distribution.
- If you know nothing about the empirical data, then you cannot assume anything with more structure than the Uniform Distribution

When we select a random deviate of a given probability density function, we are performing an integral over the PDF. The recipe to generate a random deviate is:

1. Generate a Uniform Random Deviate

2. Calculate the integral of the appropriate probability density function (PDF) from the left limit up to appropriate $x - value$ such that the area under the PDF curve is equal to the uniform random deviate in Step 1, above

3. Return the appropriate $x - value$ as the random deviate to your simulation

https://en.wikipedia.org/wiki/Uniform_distribution_(continuous)
(https://en.wikipedia.org/wiki/Uniform_distribution_(continuous))

So we iteratively solve

$$U\left[0,1\right) = \int_0^x PDF(x')\,dx'$$

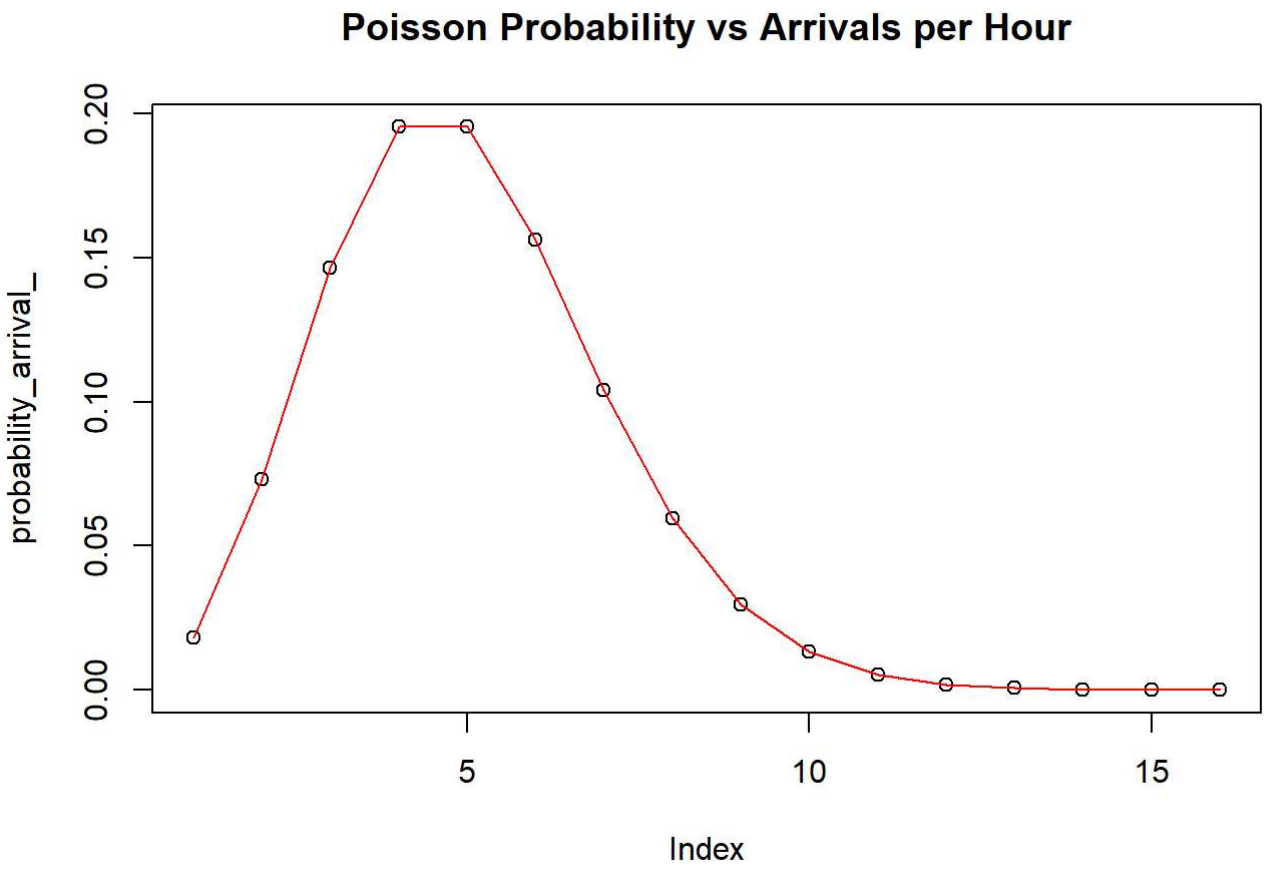
and the collection of x values comprise a distribution of random deviates of the PDF. Let's build this machinery in Base R and then we will rely on optimized libraries to do the work for us in our simulation.

Let's say we want to simulate the arrival of shipping containers to the Port of Poissionlandia, which is a make-believe world that is connected to a college campus. We know that the expectation of C containers arriving in the next hour is governed by the Poisson probability mass function with characteristic mean and variance given by λ

$$Pois\left(x \mid C\right) = \frac{e^{-\lambda}\lambda^{C}}{C!}$$

So if λ is the mean of say 4 containers per hour, then the probability mass is given by:

```
pois_ <- function(containers_, lambda_){

  (lambda_ ^ containers_) * exp(- lambda_) / factorial(containers_)


}


containers_ <- seq(0,15)

probability_arrival_ <- pois_(containers_, 4)

plot(probability_arrival_,main='Poisson Probability vs Arrivals per Hour')
lines(probability_arrival_,col="red")
```



This distribution says that given the long-term historical average of 4 Containers arriving at the port per hour, we have a 10% probability that 7 containers will arrive instead.

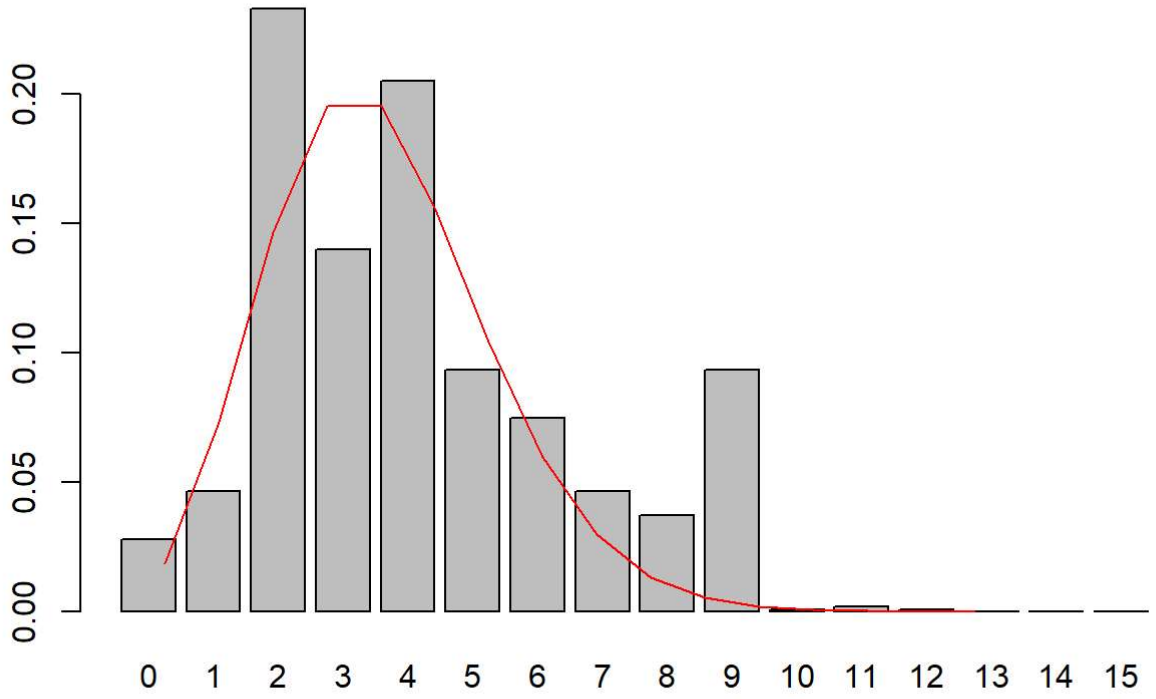
Now, for aspiring data scientist, it is very important to understand that this exercise is very dangerous. We mean, it is very dangerous to **assume** that the arrival probability density is Poisson distributed simply because it is supposed to be. Let's review the facts here:

1. We are modeling an arrival process
2. We know from school that such processes are Poisson distributed
3. We have learned through Client interviews and peer coaching on Client account knowledge that the mean number of arrivals per hour is 4
4. We drew the curve above

Where did we go wrong? Well, we were doing very well right up to Step 4, where we drew a Poisson Probability Mass Function. That was our mistake.

If we dig deeper, we might learn that the actual data is distributed like this, where the Bars are reality and the red line is the theoretical Poisson trace from above:

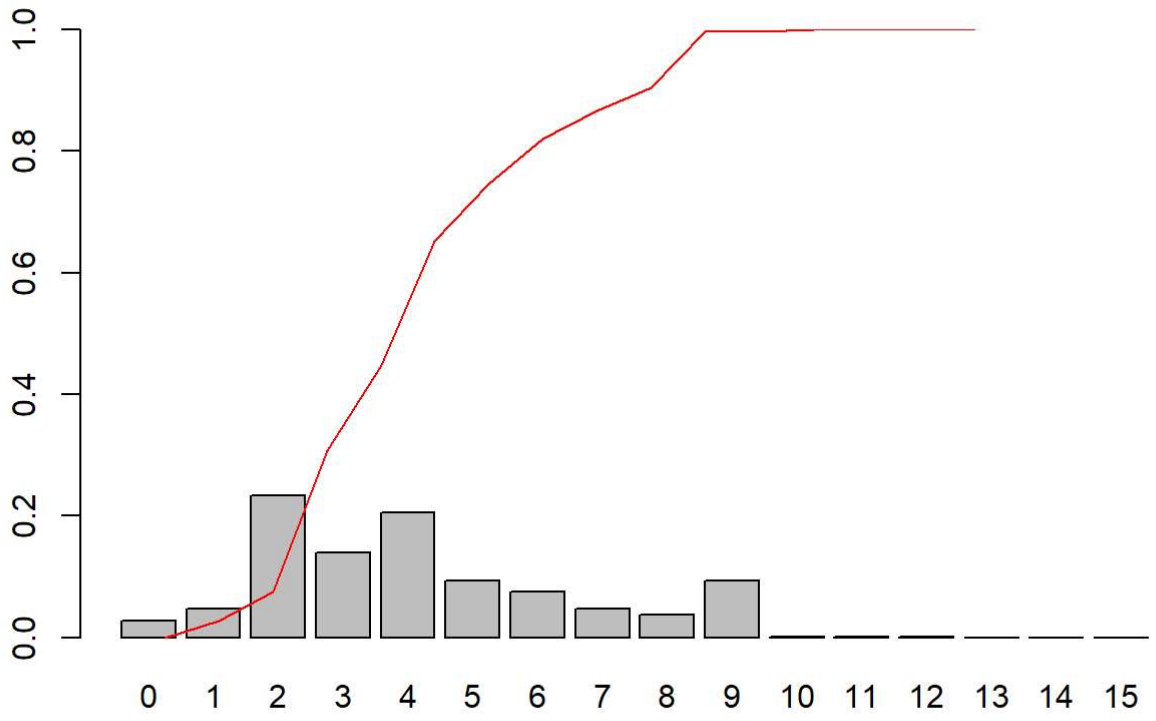
```
# Careful, this vector is Index 1 - 16, but it represents Poisson arrival probabilities for 0 -
 16 arrivals per hour

empirical_arrival_ <- c(0.03, 0.05, 0.25, 0.15, 0.22, 0.10, 0.08, 0.05, 0.04, 0.1, 0.0005, 0.002
, 0.0008, 0.00002, 0.00006, 0.000015)
empirical_arrival_ <- empirical_arrival_/sum(empirical_arrival_)

barplot(empirical_arrival_, names.arg=seq(0:15)-1)

lines(probability_arrival_,col="red")
```



The data definitely has some central tendency around 4, but there is much more to the story. Reality is not cooperating with theory, which is normal, by the way. As Industrial and Applied Mathematicians, we must generate random deviates that mimic reality, else we create risk in terms of life, limb, or money damages. Here is how we generate random deviates for an arbitrary Empirical Probability Distribution or Mass Function.

First, we integrate the Empirical Probability Mass Function to calculate the Cumulative Mass Function. There is a function in R to do that for us, but let's do it manually in Base R for illustrative value; remember, we will have to explain ourselves to our Clients. R vectors are Base 1, which introduces a bit of an accounting inconvenience.

```
# A CDF is always of length 1 longer than the PDF vector
nCDF_ <- length(empirical_arrival_) + 1

# Instantiate the CDF vector
cumulative_arrival_ <- vector("numeric", nCDF_)

# Seed the first vector element, which is always zero
cumulative_arrival_[1] <- 0.00

# Integrate the PDF and deal with the accounting offset
for(iCDF_ in 1:(nCDF_ - 1)){

  #cumulative_arrival_[iCDF_ + 1] <- cumulative_arrival_[iCDF_] + probability_arrival_[iCDF_]
  cumulative_arrival_[iCDF_ + 1] <- cumulative_arrival_[iCDF_] + empirical_arrival_[iCDF_]

}


barplot(empirical_arrival_, ylim=(0:1),names.arg=seq(1:(nCDF_ - 1))-1)
lines(cumulative_arrival_[1:16], col='red')
```



Let's check our work. The CMF always starts at zero. The first bar of the PMF is 0.0279487048, so the second entry in the CMF is the sum of the first bar, which is 0.0279487048. The third entry in the CMF is the sum of the first two bars of the PMF, 0.0279487 + 0.0465812 = 0.0745299. So the CMF seems to be working.

```
print(noquote('Empirical Arrival Function'))
```

```
## [1] Empirical Arrival Function
```

```
noquote(format(empirical_arrival_, digits=6, scientific=FALSE))
```

```
##  [1] 0.0279487048 0.0465811747 0.2329058734 0.1397435241 0.2049571686
##  [6] 0.0931623494 0.0745298795 0.0465811747 0.0372649397 0.0931623494
## [11] 0.0004658117 0.0018632470 0.0007452988 0.0000186325 0.0000558974
## [16] 0.0000139744
```

```
print(noquote(' '))
```

```
## [1]
```

```
print(noquote('Cumulative Arrival Function'))
```

```
## [1] Cumulative Arrival Function
```

```
noquote(format(cumulative_arrival_, digits=6, scientific=FALSE))
```

```
##  [1] 0.0000000 0.0279487 0.0745299 0.3074358 0.4471793 0.6521364 0.7452988
##  [8] 0.8198287 0.8664098 0.9036748 0.9968371 0.9973029 0.9991662 0.9999115
## [15] 0.9999301 0.9999860 1.0000000
```

So the Cumulative Mass Function tallies each bar of the Probability Mass Function (PMF) and sums from zero to 1.0. We can use the CMF to do a lookup to generate Empirical Random Variates for our simulation. Remember the recipe above. First, we generate a Uniform Random Variate, then integrate the Probability Density Function (PDF), which for a discrete variable means to sum the Probability Mass Function. But wait! We just did that when we calculated the Cumulative Mass Function. So the act of summing the Probability Mass Function is equivalent to doing a simple lookup against the Cumulative Mass Function.

Let's pull N Empirical Random Variates from our Cumulative Mass Function, then plot an red Simulated Mass Function bars against our gray Empirical Mass Function bars to see whether they match. Let's stay in Base R and use the which() function with a threshold. We will pull a Uniform Random Variate, then ask the R question which of the Cumulative Mass Function vector elements are less than or equal to the Uniform value. Then we will take the maximum of those indices and return that number as our Empirical Random Variate. Then we will plot them with our Bars, and our expectation is that they will match very nicely. Let's do it.

```
# Generate N Uniform Random Variants
nUnifs_ <- 100

# Create a vector to hold the collection of results
max_cdf_index_ <- vector("numeric",nUnifs_)

# Pull N Uniform Random Variates
rUnifs_ <- runif(n=nUnifs_,min=0, max=1)

# For each Uniform Random Variate, ask 'which of the Cumulative Arrival Function vector elements
are
# less than or equal to the URV'  From that vector of one or more entries, save the maximum inde
x,
# which is our Empirical Random Variate
for(iUnif_ in 1:nUnifs_){

  which_ <- which(cumulative_arrival_ <= rUnifs_[iUnif_])
  max_cdf_index_[iUnif_] <- max(which_)


}

# Remember, R is Base-1 and Poisson is Base 0, so we need to keep track of the accounting
#table_ <- table(max_cdf_index_)
table_ <- vector("numeric",nCDF_ - 1)

for(iCDF_ in 1:(nCDF_-1)){

  which_ <- which(max_cdf_index_ == iCDF_)
  table_[iCDF_] <- length(which_)

}
table_
```

```
##  [1]  4  5 28 15 19  8  9  3  0  8  0  1  0  0  0  0
```

Well, that is disappointing. We did not even sample the entire space. But let's get the machinery working so that we can scale.

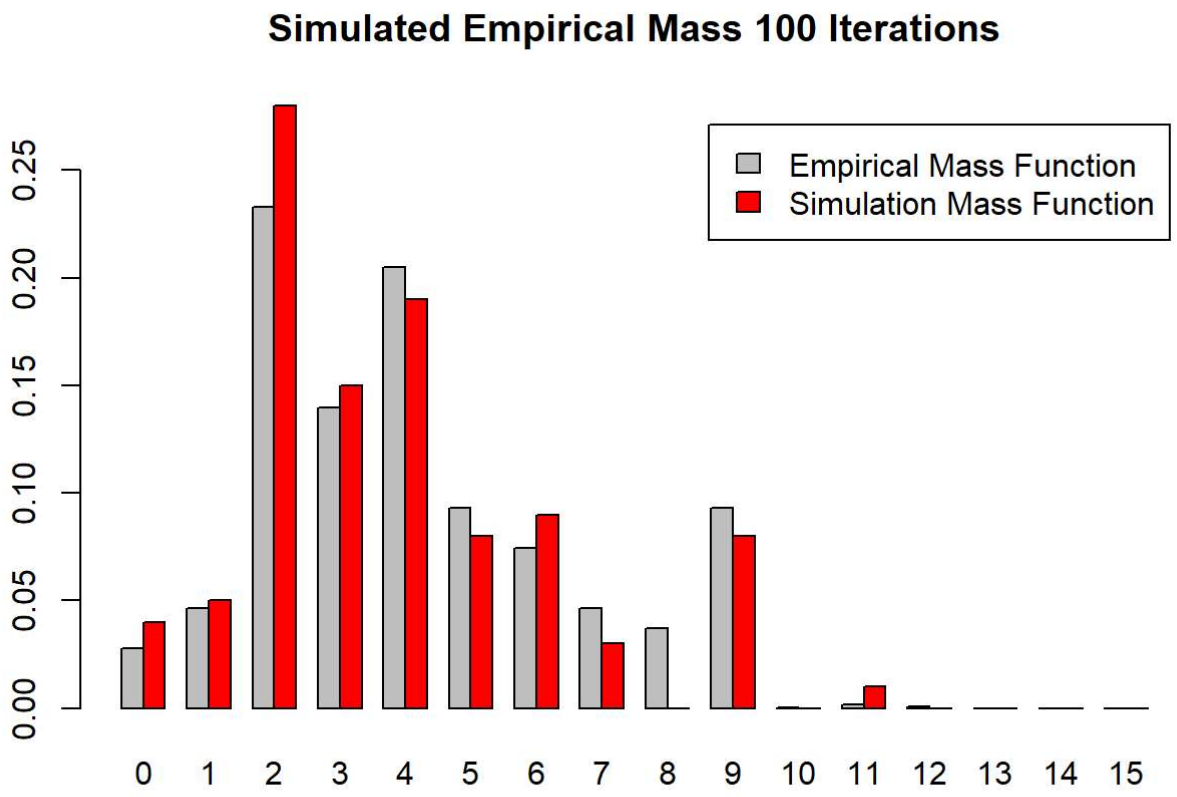
Now we have to normalize the simulation's Probability Mass Function, which we will term the Simulation Arrival Function in parallel with the Poisson Arrival Function. Remember that R vectors are 1-Based, but an Arrival Function is 0-Based, so we have to keep track of the accounting…

```
simulation_arrival_ <- table_/nUnifs_
simulation_arrival_
```

```
##  [1] 0.04 0.05 0.28 0.15 0.19 0.08 0.09 0.03 0.00 0.08 0.00 0.01 0.00 0.00 0.00
## [16] 0.00
```

Ok, let's see how we did. We need to pad our Simulation vector to make the graph work…

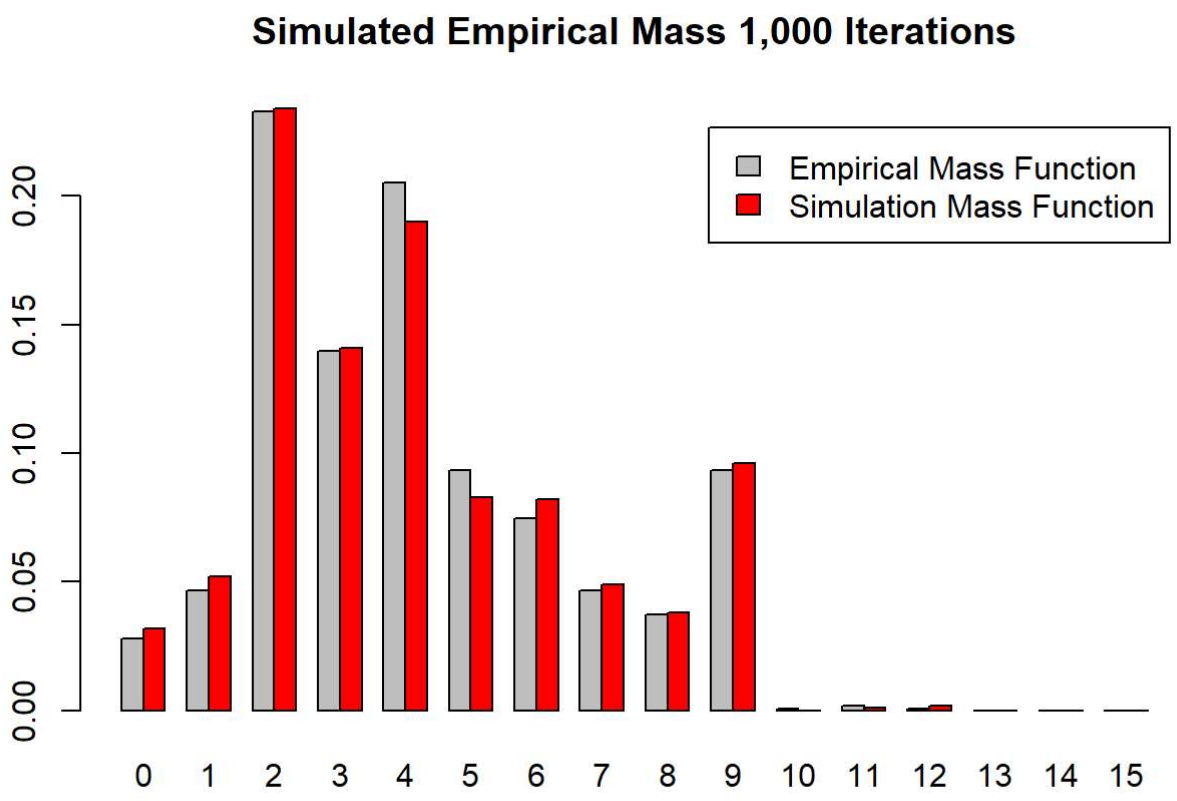
```
simulation_arrival_ <- as.vector(simulation_arrival_)

suppressWarnings(plot_ <- rbind(empirical_arrival_,simulation_arrival_))

main_ <- paste('Simulated Empirical Mass', format(nUnifs_,scientific=FALSE,big.mark=","),'Iterat
ions', sep=' ')
suppressWarnings(barplot(plot_, beside = TRUE, names.arg=seq(1:16)-1,col=c("gray","red"),main=ma
in_,legend=c('Empirical Mass Function','Simulation Mass Function')))
```



Now that is a respectible but unsatisfactory simulation. With 100 random draws, we completely failed to simulate the Empirical Probability Mass Distribution; that is, our Empirical Arrival Function is completely wrong. Let's throw more compute at the problem, which translates into more money, of course.

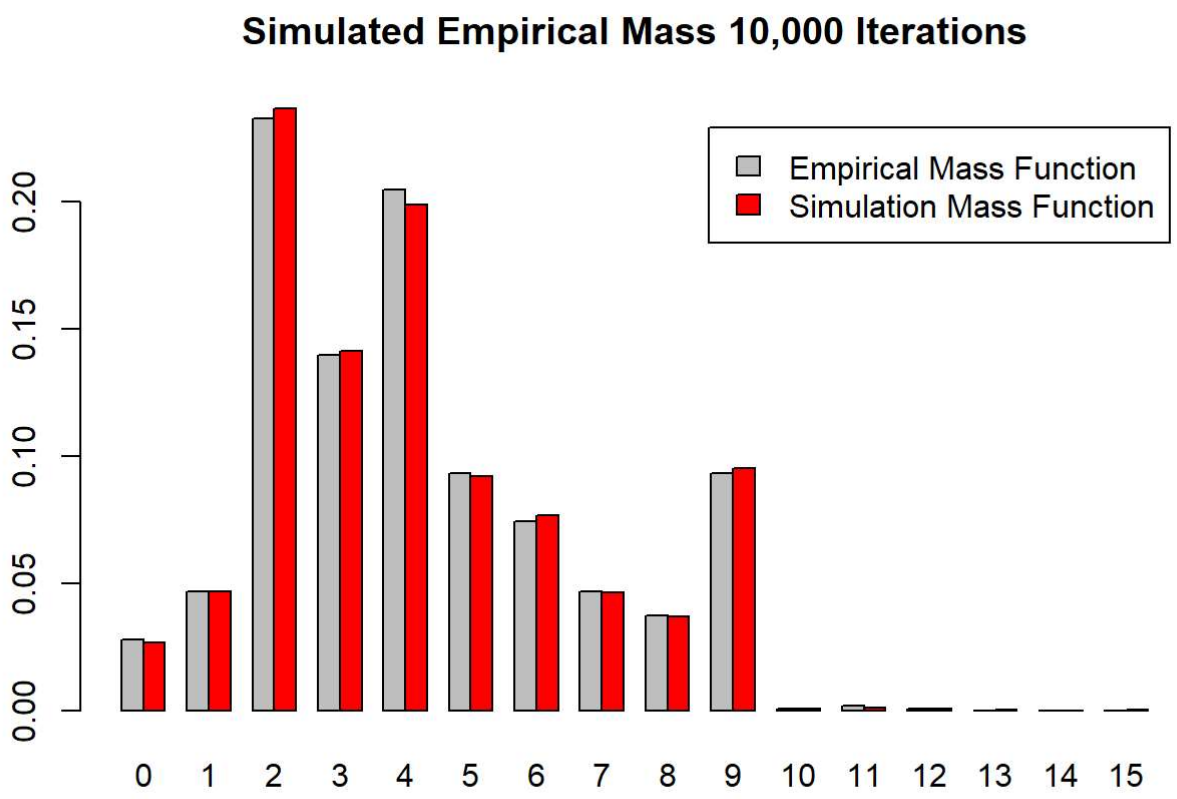
Increase to 1000 iterations.

```
nUnifs_ <- 1000
max_cdf_index_ <- vector("numeric",nUnifs_)
rUnifs_ <- runif(n=nUnifs_,min=0, max=1)

for(iUnif_ in 1:nUnifs_){

  which_ <- which(cumulative_arrival_ <= rUnifs_[iUnif_])
  max_cdf_index_[iUnif_] <- max(which_)

}


for(iCDF_ in 1:(nCDF_-1)){

  which_ <- which(max_cdf_index_ == iCDF_)
  table_[iCDF_] <- length(which_)

}

#table_
#table_ <- table(max_cdf_index_)
simulation_arrival_ <- table_/nUnifs_
simulation_arrival_ <- as.vector(simulation_arrival_)
suppressWarnings(plot_ <- rbind(empirical_arrival_,simulation_arrival_))
main_ <- paste('Simulated Empirical Mass', format(nUnifs_,scientific=FALSE,big.mark=","),'Iterat
ions', sep=' ')
suppressWarnings(barplot(plot_, beside = TRUE, names.arg=seq(0:15)-1,col=c("gray","red"),main=ma
in_,legend=c('Empirical Mass Function','Simulation Mass Function')))
```



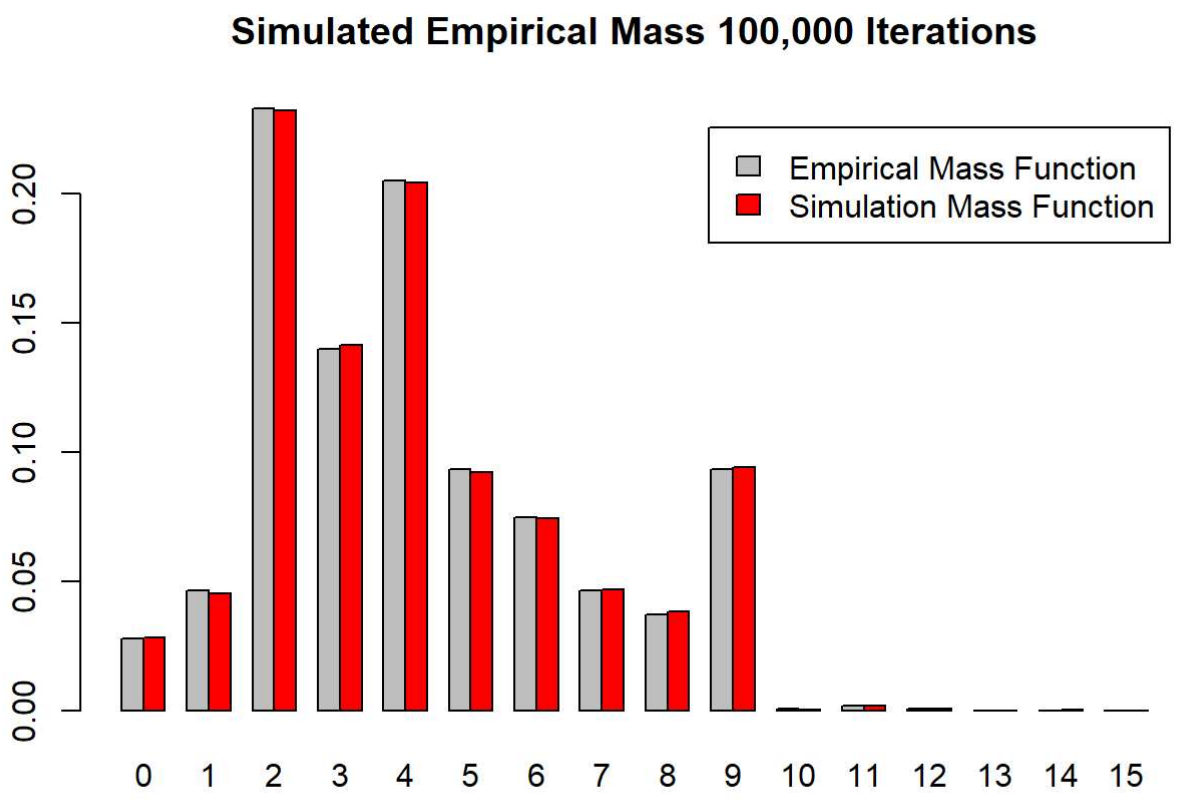
Well, it is shaping up. We have the essential physics in there, but we can do better. We seem to be simulating the more-probable C values, but we are having trouble with the less-probable ones, which makes perfect sense. With a small number of Random Draws, we have less chance to sample a small gray bar's probability density. Let's increase the number of iterations to 10,000 and try again.

```
nUnifs_ <- 10000
max_cdf_index_ <- vector("numeric",nUnifs_)
rUnifs_ <- runif(n=nUnifs_,min=0, max=1)

for(iUnif_ in 1:nUnifs_){

  which_ <- which(cumulative_arrival_ <= rUnifs_[iUnif_])
  max_cdf_index_[iUnif_] <- max(which_)

}

for(iCDF_ in 1:(nCDF_-1)){

  which_ <- which(max_cdf_index_ == iCDF_)
  table_[iCDF_] <- length(which_)

}


#table_ <- table(max_cdf_index_)
simulation_arrival_ <- table_/nUnifs_
simulation_arrival_ <- as.vector(simulation_arrival_)
suppressWarnings(plot_ <- rbind(empirical_arrival_,simulation_arrival_))
main_ <- paste('Simulated Empirical Mass', format(nUnifs_,scientific=FALSE,big.mark=","),'Iterat
ions', sep=' ')
suppressWarnings(barplot(plot_, beside = TRUE, names.arg=seq(0:15)-1,col=c("gray","red"),main=ma
in_,legend=c('Empirical Mass Function','Simulation Mass Function')))
```



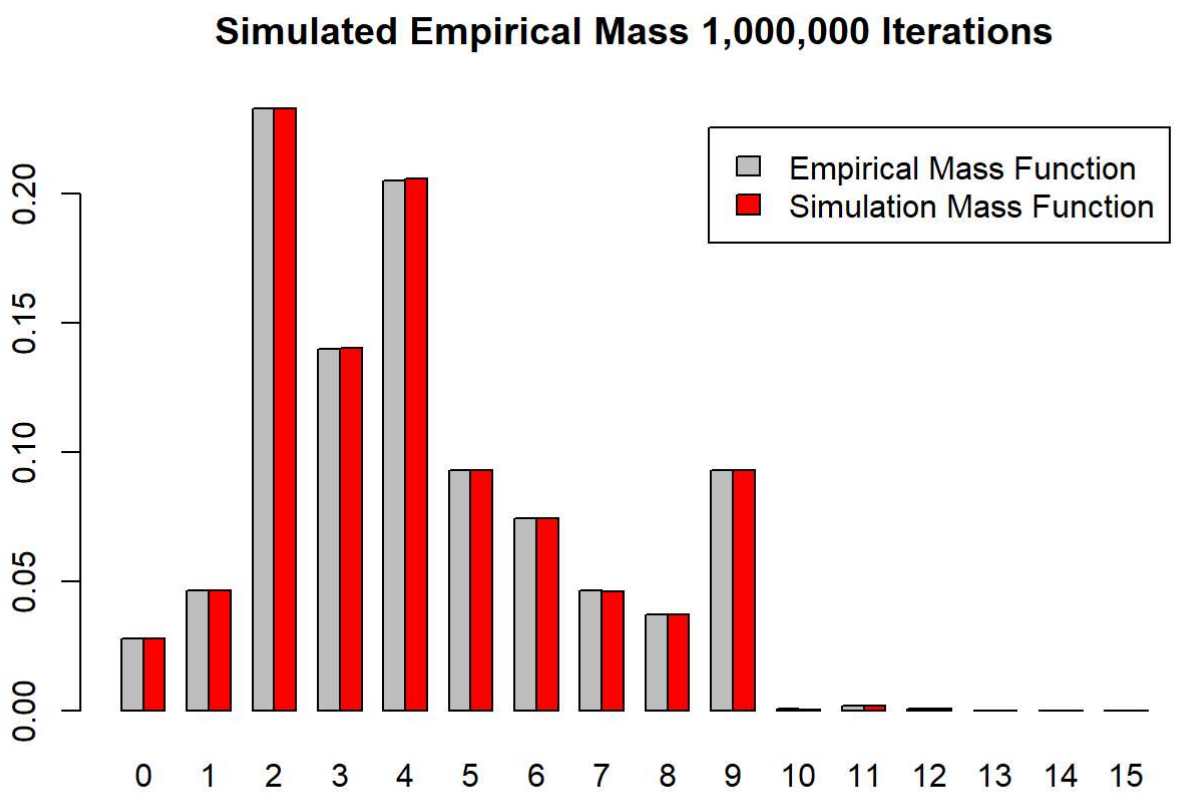
That is starting to take shape, but there is something wonky going on over to the right. Let's up the ante once again - 1-hundred-thousand iterations this time. Maybe the dice inside the Pseudo-Random Number Generator will calm down a bit…

```
nUnifs_ <- 100000
max_cdf_index_ <- vector("numeric",nUnifs_)
rUnifs_ <- runif(n=nUnifs_,min=0, max=1)

for(iUnif_ in 1:nUnifs_){

  which_ <- which(cumulative_arrival_ <= rUnifs_[iUnif_])
  max_cdf_index_[iUnif_] <- max(which_)

}


for(iCDF_ in 1:(nCDF_-1)){

  which_ <- which(max_cdf_index_ == iCDF_)
  table_[iCDF_] <- length(which_)

}


#table_ <- table(max_cdf_index_)
simulation_arrival_ <- table_/nUnifs_
simulation_arrival_ <- as.vector(simulation_arrival_)
suppressWarnings(plot_ <- rbind(empirical_arrival_,simulation_arrival_))
main_ <- paste('Simulated Empirical Mass', format(nUnifs_,scientific=FALSE,big.mark=","),'Iterat
ions', sep=' ')
suppressWarnings(barplot(plot_, beside = TRUE, names.arg=seq(0:15)-1,col=c("gray","red"),main=ma
in_,legend=c('Empirical Mass Function','Simulation Mass Function')))
```



That is starting to work. Up the spend to 1-million iterations and recompute.

```
nUnifs_ <- 1000000
max_cdf_index_ <- vector("numeric",nUnifs_)
rUnifs_ <- runif(n=nUnifs_,min=0, max=1)

for(iUnif_ in 1:nUnifs_){

  which_ <- which(cumulative_arrival_ <= rUnifs_[iUnif_])
  max_cdf_index_[iUnif_] <- max(which_)

}

for(iCDF_ in 1:(nCDF_-1)){

  which_ <- which(max_cdf_index_ == iCDF_)
  table_[iCDF_] <- length(which_)

}


#table_ <- table(max_cdf_index_)
simulation_arrival_ <- table_/nUnifs_
simulation_arrival_ <- as.vector(simulation_arrival_)
suppressWarnings(plot_ <- rbind(empirical_arrival_,simulation_arrival_))
main_ <- paste('Simulated Empirical Mass', format(nUnifs_,scientific=FALSE,big.mark=","),'Iterat
ions', sep=' ')
suppressWarnings(barplot(plot_, beside = TRUE, names.arg=seq(0:15)-1,col=c("gray","red"),main=ma
in_,legend=c('Empirical Mass Function','Simulation Mass Function')))
```



Well, let's stop there. We are still having trouble with the thin tail to the right, but that cannot be helped. This is working very nicely. We now have an Empirical Arrival Function that we can use to model the business problem's historical data. So there is a very important lesson for us here. In simulation, ***You can have as little error or as much accuracy, precision, and recall as you are willing to pay for. And there is a threshold minimum number of iterations for a Monte Carlo model to converge to nature. And even still, some regions of nature may remain inaccessible due to the nature of distributions of Uniform Random Draws over a finite domain.***

After all, the Empirical Arrival Function for $C = 15$ is 1.397435e-05, which is 0.00001397435 or 14 chances in 1-million Draws or 1.4 Draws in 100,000 Tries. So we actually did OK. Here is the Empirical and Simulated Arrival Functions. You can see that we matched very nicely. It is very tight on the left, but in the thin tail, it wanders a bit since that probability space is not as accessible:

```
print(noquote('Empirical Arrival Function'))
```

```
## [1] Empirical Arrival Function
```

```
noquote(format(empirical_arrival_, digits=3, scientific=FALSE))
```

```
##  [1] 0.0279487 0.0465812 0.2329059 0.1397435 0.2049572 0.0931623 0.0745299
##  [8] 0.0465812 0.0372649 0.0931623 0.0004658 0.0018632 0.0007453 0.0000186
## [15] 0.0000559 0.0000140
```

```
print(noquote(' '))
```

```
## [1]
```

```
print(noquote('Simulated Arrival Function'))
```

```
## [1] Simulated Arrival Function
```

```
noquote(format(simulation_arrival_, digits=3, scientific=FALSE))
```

```
##  [1] 0.027894 0.046285 0.233059 0.140252 0.205803 0.093141 0.074202 0.046206
##  [9] 0.037230 0.092805 0.000441 0.001860 0.000737 0.000018 0.000053 0.000014
```

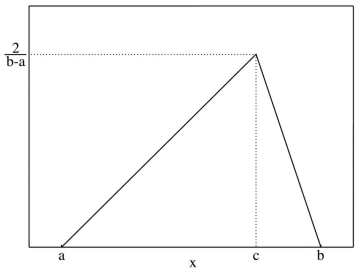
The Triangular Distribution

Remember the rules of thumb about simulation; we never want to assume too much, and if we know nothing about the domain, we should use Uniform Random Variants, but of course, that is unsatisfying on the one hand and unnecessary on the other. Even the most cursory of literature reviews or expert interviews will reveal something about the problem. So if it is disingenuous to claim say Gaussian behavior, which requires us to know something about the mean and variance around the mean, or unsatisfying to rely on Uniform distributions throughout, what then?

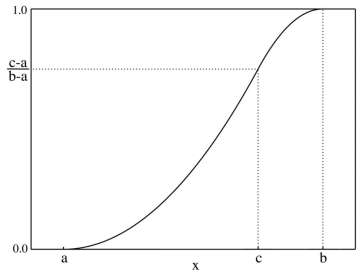
In this case, the Triangular distribution is invaluable. The Triangular distribution possessed many admirable qualities for our purposes. It is bounded both on the left and right, so it cannot run away from us with a crazy outlier that is unphysical for our problem set. And it does not require us to make unsubstantiated claims about our understanding of the distribution. Further, Client or Peer subject-matter experts can most always help us understand the range and central tendency of an empirical distribution; that is, we can find someone with expert knowledge of the business problem to tell us something of the nature “the number of containers arriving per hour is somewhere between zero and 10, with the most-common number being 3 or 4.”

This type of utterance can be modeled very well with a Triangular distribution, which is parameterized with the values $a, b, \& \, c$, where a is the left bound, b is the right bound, and c is the most-likely value.

The Triangular distribution has Probability Density Function:



…and Cumulative Density Function



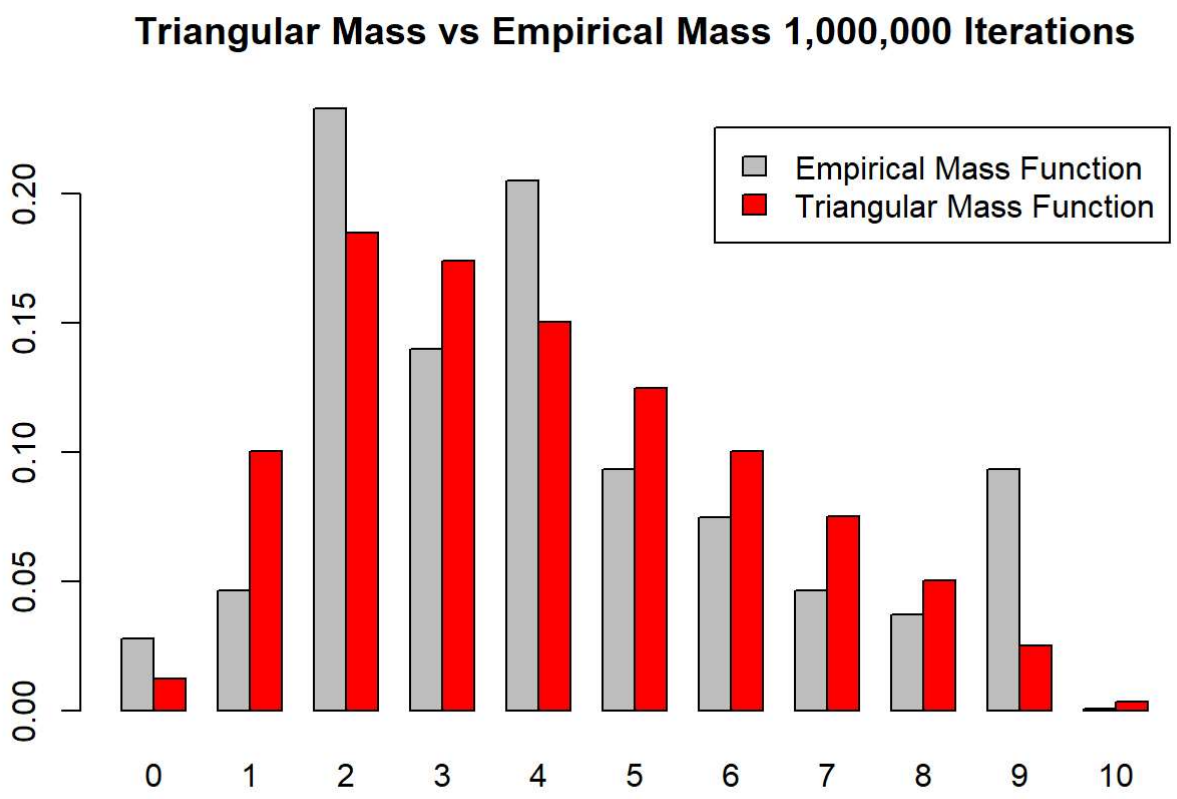
Let's model our Container arrival problem with a Triangular distribution using R's Triangle library. We will simulate 1-million Triangular random variates, bin them, and plot the resulting distribution against our empirical mass function. Let's take the most-common value, C to be 2 since we have to manipulate our Triangle to fit the Empirical peaks at 2 and 4. The trick is to have the minimum error, and we can measure that.

```
# Draw N double-precision Triangular Random Variates
rtriangle_ <- rtriangle(n= nUnifs_, a=0, b=10, c=2)

# Round to integer values
rtriangle_ <- round(rtriangle_, digits=0)

# Bin and table the integers
table_ <- table(rtriangle_)
table_
```

```
## rtriangle_
##      0      1      2      3      4      5      6      7      8      9     10
##  12431 100125 184759 174063 150372 124642 100235  75021  50123  25063   3166
```

Now normalize the distribution of Triangular Random Variates to create a Triangular Probability Mass Function, and plot against our Empirical Mass Function.

```
# Extract the numeric values of the Table object
triangular_arrival_ <- as.vector(table_)

# Normalize the vector
triangular_arrival_ <- triangular_arrival_ / nUnifs_

# Plot the Triangular and Empirical Mass Functions
suppressWarnings(plot_ <- rbind(empirical_arrival_[1:11],triangular_arrival_))
main_ <- paste('Triangular Mass vs Empirical Mass', format(nUnifs_,scientific=FALSE,big.mark=","
),'Iterations',sep=' ')
suppressWarnings(barplot(plot_, beside = TRUE, names.arg=seq(0:10)-1,col=c("gray","red"),main=ma
in_,legend=c('Empirical Mass Function','Triangular Mass Function')))
```



To get a feel for how we did, let's calculate a Mean Absolute Percentage Error (MAPE) between our Triangular and Empirical Masses.

```
# [M]ean [A]bsolute [P]ercentage [E]rror

# Calculate the error
E_ <-  triangular_arrival_ - empirical_arrival_[1:11]

# Take absolute value
A_ <- abs(E_)

# Calculate the absolute error as a percent of the Empirical Mass
P_ <- A_ / empirical_arrival_[1:11]

# Average the percentage errors to obtain a Mean Absolute Percentage Error (MAPE)
M_ <- mean(P_)
M_
```

```
## [1] 0.9626753
```

So our mean error is over 90%. Goodness, that is terrible! But compared to what? Let's check the MAPE of the theoretical Poisson Mass Function.

```
# [M]ean [A]bsolute [P]ercentage [E]rror
E_ <-  probability_arrival_[1:11] - empirical_arrival_[1:11]
A_ <- abs(E_)
P_ <- A_ / empirical_arrival_[1:11]
M_ <- mean(P_)
M_
```

```
## [1] 1.318909
```

Whoa! The error for a theoretical Poisson Arrival Function is over 130% That is really horrible. How can the theory be so wrong? Well, the theory is seldom ever correct. Your professor probably forgot to mention that in school!

Remember, neither of these techniques are superior to the Empirical Mass Function, but if we just want to get the Multiphysics working so that we can get into our problem, the Triangular Mass Function is the most dexterous of all. It does not extract disingenuous assumptions from us for the privilege of using it. We do not have to claim that we know a mean or standard deviation. Further, we can interview subject matter experts who can impart domain knowledge and their narrative can be encoded into the Triangular distributions' parameters very quickly.

We do give up some MAPE, but we get a quick time-to-impact for our trouble. We cannot stop with the Triangular Distribution, but it is a great place to start.

Hey! Let's check the MAPE of our Simulation Arrival Mass that we worked so hard to model.

```
# [M]ean [A]bsolute [P]ercentage [E]rror
E_ <-  simulation_arrival_[1:11] - empirical_arrival_[1:11]
A_ <- abs(E_)
P_ <- A_ / empirical_arrival_[1:11]
M_ <- mean(P_)
M_
```

```
## [1] 0.007950928
```

Bam! Only a little over 30 basis points! Four tenths of 1 percent! Nothing beats Simulated Empirical Random Variants! But again, if the Client cannot produce the historical data, and you want to get the simulation working, the Triangular Distribution is a good place to start.

Monte Carlo Simulation of SEIR

Simulation of SEIR starts with gathering the WHO data. As in the prior paper, we will examine Guinea. We must simulate σ: the Average Frequency of Incubation, γ, the Average Frequency of Infectiousness, β_0, the Rate of Transmission in the absence of Public Health Control Intervention Measures, f: the Fatality Rate, κ the exponential decay constant for $\beta(t)$, the time-dependent Transmission Rate in the presence of Public Health Control Intervention Measures.

We will calculate MAPE, which will guide us to candidate combinations of the model parameters. The recipe is:

1. For each parameter, select a Probability Distribution or Mass Function
2. Load vectors of random variants for each parameter
3. Calculate SEIR trajectories for each iteration
4. Measure MAPE

5. Plot MAPE
6. Select the minimum MAPE
7. Examine the parameters that conspired to produce the minimum MAPE and test for reasonableness

The SEIR calculation was discussed in the prior paper, so we will focus on the Monte Carlo simulation here.

https://4-vector.org/2020/04/27/4vector-org-2020-04-25-2100-mai/ (https://4-vector.org/2020/04/27/4vector-org-2020-04-25-2100-mai/)

First, let's look at our data from the Althaus paper.

https://4vector.files.wordpress.com/2020/04/althaus_west-africa.pdf
(https://4vector.files.wordpress.com/2020/04/althaus_west-africa.pdf)

The data has been extracted and compiled here:

https://4vector.files.wordpress.com/2020/04/ebola_2013.csv_.pdf
(https://4vector.files.wordpress.com/2020/04/ebola_2013.csv_.pdf)

```
file_ <- '..\\csv\\ebola_2013.csv'
df_csv_ <- read.csv(file=file_,stringsAsFactors = FALSE,header=TRUE,sep=',')
str(df_csv_)
```

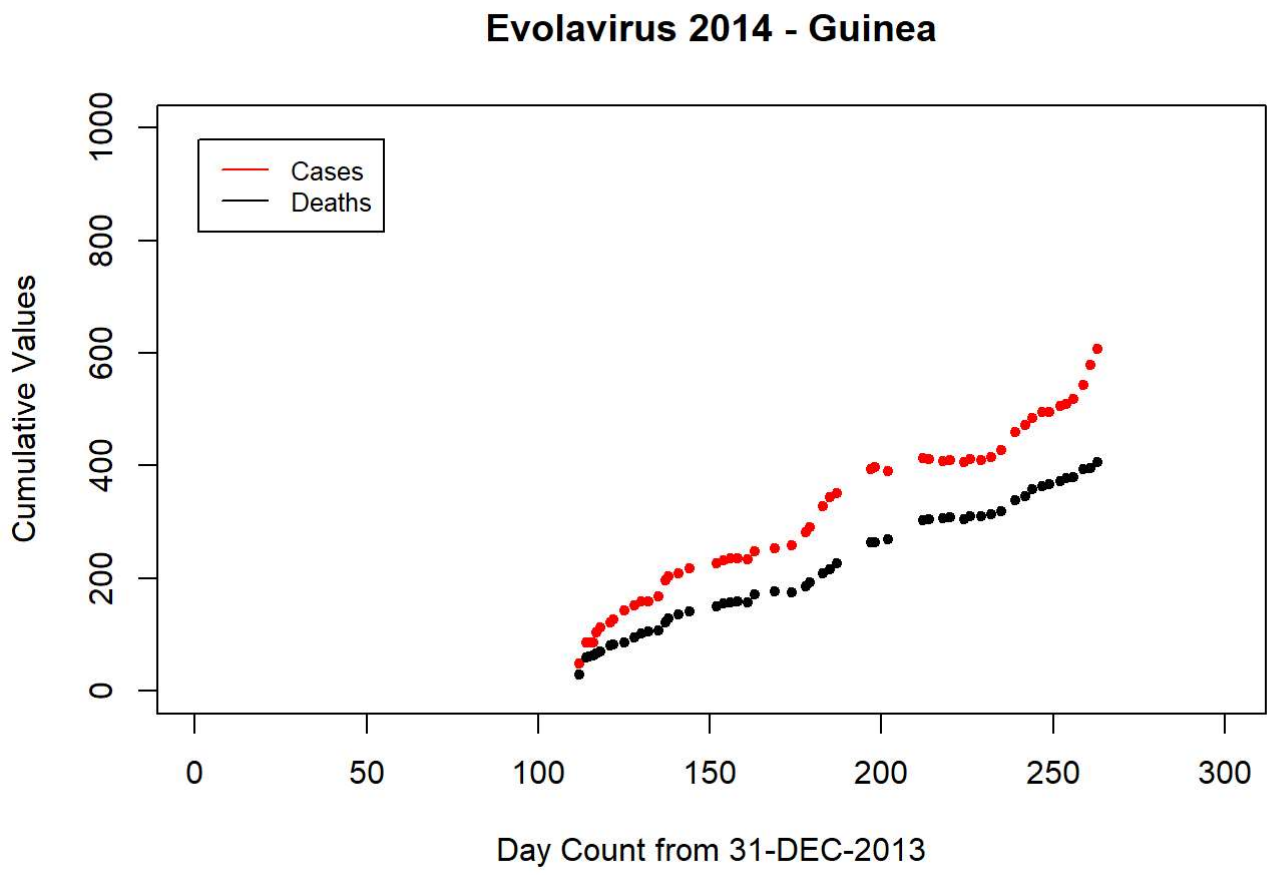
```
## 'data.frame':    300 obs. of  8 variables:
##  $ index     : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ yyyymmdd  : int  NA NA NA NA NA NA NA NA NA NA ...
##  $ cases_g_  : int  NA NA NA NA NA NA NA NA NA NA ...
##  $ deaths_g_ : int  NA NA NA NA NA NA NA NA NA NA ...
##  $ cases_sl_ : int  NA NA NA NA NA NA NA NA NA NA ...
##  $ deaths_sl_: int  NA NA NA NA NA NA NA NA NA NA ...
##  $ cases_l_  : int  NA NA NA NA NA NA NA NA NA NA ...
##  $ deaths_l_ : int  NA NA NA NA NA NA NA NA NA NA ...
```

Recall from the prior paper that the West-Africa data is sparse. The data contain Cases and Deaths for Guinea, Sierra Leone, and Liberia. Be careful not to impute data; blanks are not zero. You can imagine that in the heat of a public-health battle with Ebolavirus, that one might become a bit too busy to tabulate data, so because the data is sparse, the actual arrival function for Cases and Deaths might have to be modeled as a Poisson Arrival Process. As amply demonstrated above, actual Empirical Mass is superior to theoretical Poisson Mass, but query whether Empirical Mass is available during an Ebolavirus outbreak.

In the first paper, we neglected this subtlety and treated Deaths and Cases as continuous variables, which is preposterous since you cannot have 0.23 of a Death, but again, the essential Multiphysics of the SEIR trajectories do not depend on this erudite point for the purposes of this primer.

For plotting purposes, replace the NULLS with blanks. Extract Guinea Cases and Deaths for downstream analysis and make note of which data were non-blank. Let's plot the data on a blank canvas for a visual, then we will undertake simulation to fit the data.

```
df_csv_['cases_g_']    <- ifelse(df_csv_['cases_g_']   == 0,'',df_csv_['cases_g_']   )
df_csv_['deaths_g_']   <- ifelse(df_csv_['deaths_g_']  == 0,'',df_csv_['deaths_g_']  )
df_csv_['cases_sl_']   <- ifelse(df_csv_['cases_sl_']  == 0,'',df_csv_['cases_sl_']  )
df_csv_['deaths_sl_']  <- ifelse(df_csv_['deaths_sl_'] == 0,'',df_csv_['deaths_sl_'] )
df_csv_['cases_l_']    <- ifelse(df_csv_['cases_l_']   == 0,'',df_csv_['cases_l_']   )
df_csv_['deaths_l_']   <- ifelse(df_csv_['deaths_l_']  == 0,'',df_csv_['deaths_l_']  )

cases_g_ <- df_csv_['cases_g_']
deaths_g_ <- df_csv_['deaths_g_']

which_is_not_na_ <- which(!is.na(cases_g_))

x_ <- seq(1:300)
y_0_ <- vector("numeric",300)
y_0_[1:300] <- NA
plot(x_, y_0_, type='l', ylim=c(0,1000),xlab='Day Count from 31-DEC-2013',ylab='Cumulative Value
s',col='red',main='Evolavirus 2014 - Guinea')
points(x_, cases_g_, type="p", pch=20, col='red')
points(x_,deaths_g_, type="p", pch=20,col='black')
legend(1, 1000 - 20, legend=c("Cases", "Deaths"),col=c("red", "black"), lty=1:1, cex=0.8)
```

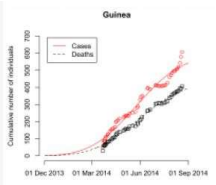


So for Guinea, Cases and Deaths were of the magnitude 600 and 400, respectively, and climbed with a characteristic SEIR trajectory.

For this simulation exercise, we will use Althaus as a “domain expert,” and treat his paper as business-analysis notes that inform our simulation.

Our task is to use such domain knowledge to parameterize the SEIR Rates and Coupling Constants with informed Simulation Mass Functions, calculate the trajectories for each set of Simulation Random Deviates, calculate the MAPE for each resulting trajectory, then use minimum MAPE to choose candidate Rates and Coupling Constants. By fitting the SEIR trajectories to the WHO data, we will calculate the SEIR Rates and Coupling Constants, including the all-important Effective Reproduction Constant.

Our goal is to simulate Althaus's findings. Here is his plot from the 2013 Ebolavirus outbreak in Guinea.



We do not have any domain knowledge, so let's "interview" Althaus and let him coach us on the magnitude of the Rates and Coupling Constants. Then we'll use Triangular Probability Distribution Functions to draw Triangular Random Variates for each variable. Remember, the Triangular Distribution gives us a good percent of the SEIR Multiphysics without requiring us to assert or assume that we know more about the statistical distributions than we actually know. In the code below, the #/N/ footnote annotations can be found in the prior paper and are left intact here for that explicit reference. We have also left Althaus's constants for illustrative reference when we override his values with a Random Variate for simulation. Remember, our quest is to "discover" Althaus's values; that is, where Althaus reported the resulting values from his study in his paper, we are going to play a game that we have only the WHO data as we pretend that the Ebolavirus epidemic is emerging, and using Althaus as a subject-matter expert (SME) on historic Ebolavirus epidemics, we will "interview" Althaus and use his wise counsel on the order-of-magnitude of the SEIR Rates and Coupling Constants to seed and bound our Probability Mass and Distribution Functions. Then by simulation, MAPE, and thresholding, we will "discover" or "replicate" Althaus's Rates and Constants.

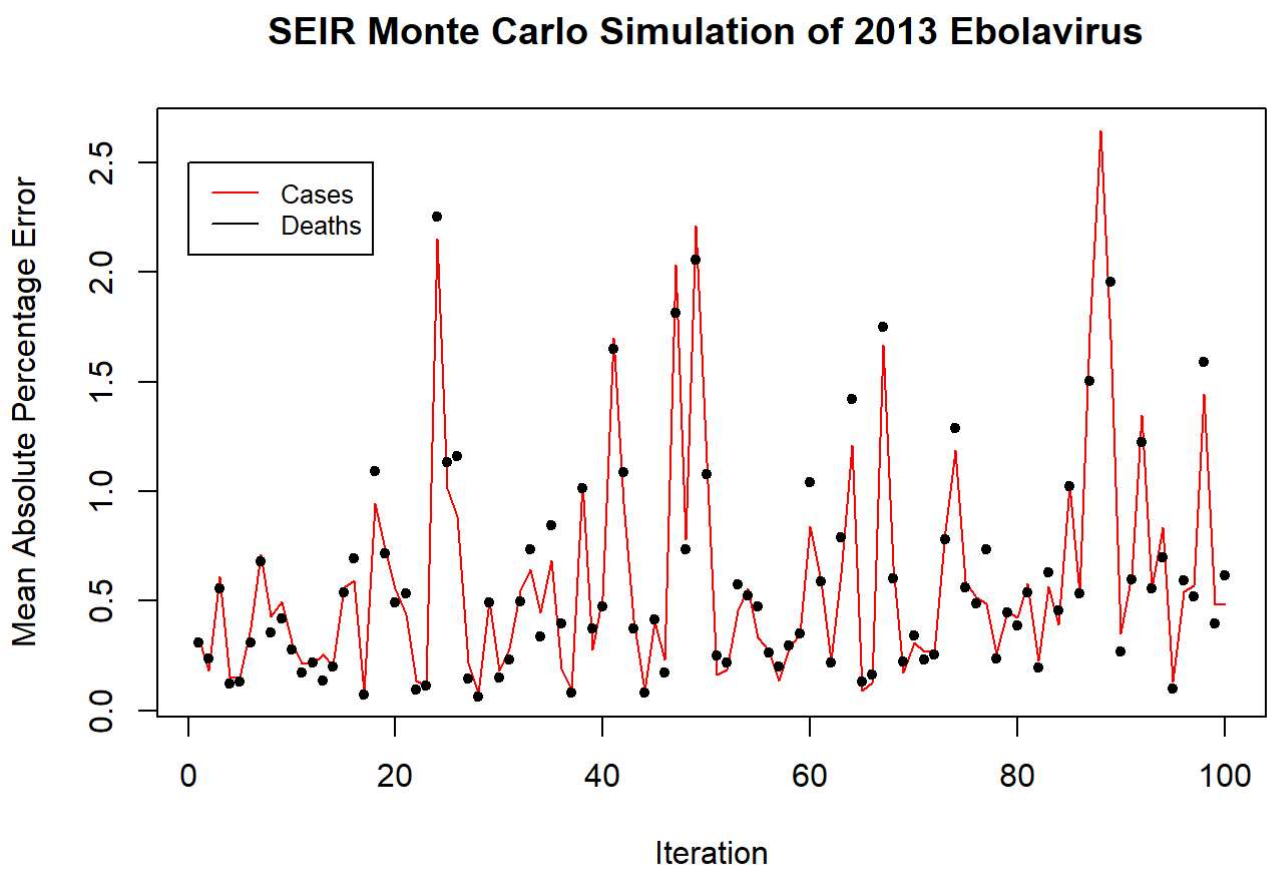
In this way, by using the Althaus paper as our "answer sheet," we will build out a powerful SEIR simulation engine that can be applied to any contagion that follows the SEIR Compartmental Epidemiology Model.

```
############################################################
nSimulations_ <- 100 #<-- SET THIS PARAMETER
############################################################


# Load Althaus's values into a data frame for tabular comparison with our simulated values below
# --------------------------------
df_althaus_ <- data.frame(
  B0=as.numeric(),
  SIGMA=as.numeric(),
  GAMMA=as.numeric(),
  KAPPA=as.numeric(),
  TAU=as.numeric(),
  F=as.numeric(),
  stringsAsFactors=FALSE
)

df_althaus_[1,'B0']    <- 0.2700
df_althaus_[1,'SIGMA'] <- 1/5.3000
df_althaus_[1,'GAMMA'] <- 1/5.6100
df_althaus_[1,'KAPPA'] <- 0.0023
df_althaus_[1,'TAU']   <- 10  # derived in our first paper
df_althaus_[1,'F']     <- 0.7400
# --------------------------------


# Random Number Factory

set.seed(137)

#B_0_ <- 0.2700  # PER DAY              #/3/
B_0_ <- rtriangle(n=nSimulations_,a=0.2500,b=0.3000,c=0.2750)


#SIGMA_ <- 1 / 5.3000                  #/5/
SIGMA_ <- rtriangle(n=nSimulations_,a=1/7.5, b=1/5.0, c=1/6.0)


#GAMMA_ <- 1 / 5.6100                  #/6/
GAMMA_ <- rtriangle(n=nSimulations_, a=1/6.0, b=1/5.0, c=1/5.5)


#K_ <- 0.0023                          #/7/
K_ <- rtriangle(n=nSimulations_, a=0.0020, b=0.0030, c=0.0025)


# Not random. This is the public-health response after the first case is recorded
TAU_ <- 10                            #/8/


#F_ <- 0.74                            #/15/
F_ <- rtriangle(n=nSimulations_,a=0.60, b=0.80, c=0.70)
```

```
df_monte_carlo_ <- data.frame(
  Iteration=as.numeric(),
  B0=as.numeric(),
  SIGMA=as.numeric(),
  GAMMA=as.numeric(),
  KAPPA=as.numeric(),
  TAU=as.numeric(),
  F=as.numeric(),
  MAPE_C=as.numeric(),
  MAPE_D=as.numeric(),
  stringsAsFactors=FALSE
)



for(iSimulation_ in 1:nSimulations_){


  df_monte_carlo_[iSimulation_,'Iteration'] <- iSimulation_
  df_monte_carlo_[iSimulation_,'B0'] <- B_0_[iSimulation_]
  df_monte_carlo_[iSimulation_,'SIGMA'] <- SIGMA_[iSimulation_]
  df_monte_carlo_[iSimulation_,'GAMMA'] <- GAMMA_[iSimulation_]
  df_monte_carlo_[iSimulation_,'KAPPA'] <- K_[iSimulation_]
  df_monte_carlo_[iSimulation_,'TAU'] <- TAU_
  df_monte_carlo_[iSimulation_,'F'] <- F_[iSimulation_]


#Create a data frame to store parameters, run-time variable values, and resulting analytic value
s.

df_simulation_ <- data.frame(
  T=as.numeric(),
  B_t=as.numeric(),
  dS=as.numeric(),
  dE=as.numeric(),
  dI=as.numeric(),
  dR=as.numeric(),
  dC=as.numeric(),
  dD=as.numeric(),
  S=as.numeric(),
  E=as.numeric(),
  I=as.numeric(),
  R=as.numeric(),
  C=as.numeric(),
  D=as.numeric(),
  R_e=as.numeric(),
  stringsAsFactors=FALSE
)


# Intitalize the model parameters.
```

```
# There is no Multiphysics here. John von Neumann said "With 4 adjustable parameters,
# I can draw an elephant; with five, I can wiggle his trunk."  This is that.

# In other words, there is nothing fundamental about this model.
# It converges at the pleasure of the adjustable parameters.


N_0_ <- 10^6                          #/1/
N_ <- N_0_                            #/2/


B_t_ <- 0                             #/4/

# Initialize the SEIR variables proper
S_ <- N_0_                            #/9/
E_ <- 0                               #/10/
I_ <- 1                               #/11/
R_ <- 0                               #/12/
D_ <- 0                               #/13/
C_ <- 1                               #/14/



nT_ <- 300                                                    #/1/
dT_ <- 1                                                      #/2/

for(iT_ in 1:nT_){

  N_ <- N_0_ - D_                                                                 #/3/
  B_t_ <- FUNC_B_t_(B_0_[iSimulation_], iT_, K_[iSimulation_], TAU_)              #/4/

  dS_ <- - B_t_ * S_ * I_  * dT_ / N_                                             #/5/
  dE_ <- (B_t_ * S_ * I_ * dT_ / N_) - (SIGMA_[iSimulation_] * E_ * dT_)          #/6/
  dI_ = (SIGMA_[iSimulation_] * E_ * dT_) - (GAMMA_[iSimulation_] * I_ * dT_)     #/7/
  dR_ <- (1 - F_[iSimulation_]) * GAMMA_[iSimulation_] * I_ * dT_                 #/8/
  dC_ <- SIGMA_[iSimulation_] * E_ * dT_                                          #/9/
  dD_ <- F_[iSimulation_] * GAMMA_[iSimulation_] * I_ * dT_                       #/10/

  S_ <- S_ + dS_                                              #/11/
  E_ <- E_ + dE_                                              #/12/
  I_ <- I_ + dI_                                              #/13/
  R_ <- R_ + dR_                                              #/14/
  C_ <- C_ + dC_                                              #/15/
  D_ <- D_ + dD_                                              #/16/

  R_e_ <- (B_t_ * S_) / (GAMMA_[iSimulation_] * N_)           #/17/


  # Load the Data Frame for this Time Step
  df_simulation_[iT_,'T']   <- iT_                            #/18/
  df_simulation_[iT_,'B_t'] <- B_t_                           #/19/

  df_simulation_[iT_,'dS']  <- dS_                            #/20/
  df_simulation_[iT_,'dE']  <- dE_                            #/21/
```

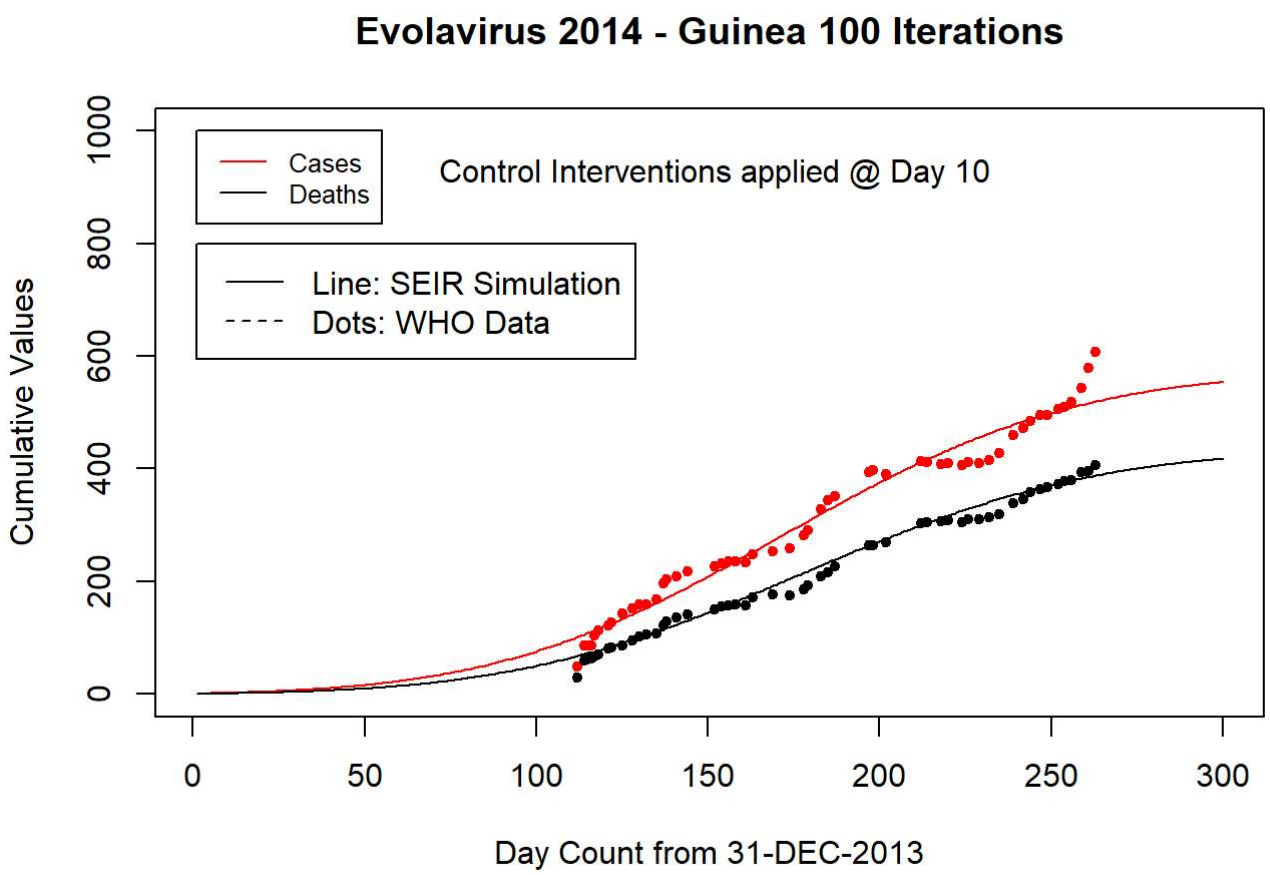
```
  df_simulation_[iT_,'dI']  <- dI_                          #/22/
  df_simulation_[iT_,'dR']  <- dR_                          #/23/
  df_simulation_[iT_,'dC']  <- dC_                          #/24/
  df_simulation_[iT_,'dD']  <- dD_                          #/25/

  df_simulation_[iT_,'S']   <- S_                           #/26/
  df_simulation_[iT_,'E']   <- E_                           #/27/
  df_simulation_[iT_,'I']   <- I_                           #/28/
  df_simulation_[iT_,'R']   <- R_                           #/29/
  df_simulation_[iT_,'C']   <- C_                           #/30/
  df_simulation_[iT_,'D']   <- D_                           #/31/

  df_simulation_[iT_,'R_e'] <- R_e_                         #/32/


}


df_monte_carlo_[iSimulation_,'MAPE_C'] <- mean(abs(df_simulation_[which_is_not_na_,'C'] - cases_
g_[which_is_not_na_])/df_simulation_[which_is_not_na_,'C'])
df_monte_carlo_[iSimulation_,'MAPE_D'] <- mean(abs(df_simulation_[which_is_not_na_,'D'] - deaths
_g_[which_is_not_na_])/df_simulation_[which_is_not_na_,'D'])


} #for


# Save the Simulation data frame as a comma-separated file (CSV) to disk for analysis
file_ <- '..\\csv\\df_monte_carlo_out.csv'

write.csv(x=df_monte_carlo_,file=file_,row.names=FALSE)


# For illustrative purposes, show this plot for small N of say 100, but the actual simulation wi
ll be run for say 1,000,000 iterations
if(nSimulations_ <= 100){
  x_ <- seq(1:nSimulations_)
  main_ <- 'SEIR Monte Carlo Simulation of 2013 Ebolavirus '
  xlab_ <- 'Iteration'
  ylab_ <- 'Mean Absolute Percentage Error'
  plot(x_, df_monte_carlo_[,"MAPE_C"], type="l", pch=20, col='red', main=main_, xlab=xlab_, ylab
=ylab_)
  points(x_,df_monte_carlo_[,"MAPE_D"], type="p", pch=20,col='black')
  legend(0, 2.5, legend=c("Cases", "Deaths"),col=c("red", "black"), lty=1:1, cex=0.8)
}
```



Now plot the solution for study. We first ask the R question, “Which MAPE is the smallest?” The R which() function returns the index of the MAPE vector element with the smallest value, and we can use that index throughout to extract each Rate and Coupling Constant to seed the integration of the SEIR equations.

```
min_d_ <- min(df_monte_carlo_[,'MAPE_D'])

which_ <- which(df_monte_carlo_[,'MAPE_D'] == min_d_)



# Intitalize the model parameters.

# There is no Multiphysics here. John von Neumann said "With 4 adjustable parameters,
# I can draw an elephant; with five, I can wiggle his trunk."  This is that.

# In other words, there is nothing fundamental about this model.
# It converges at the pleasure of the adjustable parameters.


N_0_ <- 10^6                          #/1/
N_ <- N_0_                            #/2/

B_0_ <- df_monte_carlo_[which_,"B0"]  # PER DAY             #/3/
B_t_ <- 0                                                  #/4/
SIGMA_ <- df_monte_carlo_[which_,'SIGMA']                  #/5/
GAMMA_ <- df_monte_carlo_[which_,'GAMMA']                  #/6/

K_ <- df_monte_carlo_[which_,"KAPPA"]                      #/7/


TAU_ <- df_monte_carlo_[which_,"TAU"]                      #/8/

# Initialize the SEIR variables proper
S_ <- N_0_                            #/9/
E_ <- 0                               #/10/
I_ <- 1                               #/11/
R_ <- 0                               #/12/
D_ <- 0                               #/13/
C_ <- 1                               #/14/
F_ <- df_monte_carlo_[which_,"F"]                          #/15/


nT_ <- 300                                                 #/1/
dT_ <- 1                                                   #/2/

for(iT_ in 1:nT_){

  N_ <- N_0_ - D_                                          #/3/
  B_t_ <- FUNC_B_t_(B_0_, iT_, K_, TAU_)                   #/4/

  dS_ <- - B_t_ * S_ * I_  * dT_ / N_                      #/5/
  dE_ <- (B_t_ * S_ * I_ * dT_ / N_) - (SIGMA_ * E_ * dT_)   #/6/
  dI_ = (SIGMA_ * E_ * dT_) - (GAMMA_ * I_ * dT_)          #/7/
  dR_ <- (1 - F_) * GAMMA_ * I_ * dT_                      #/8/
  dC_ <- SIGMA_ * E_ * dT_                                 #/9/
  dD_ <- F_ * GAMMA_ * I_ * dT_                            #/10/
```

```
  S_ <- S_ + dS_                                          #/11/
  E_ <- E_ + dE_                                          #/12/
  I_ <- I_ + dI_                                          #/13/
  R_ <- R_ + dR_                                          #/14/
  C_ <- C_ + dC_                                          #/15/
  D_ <- D_ + dD_                                          #/16/

  R_e_ <- (B_t_ * S_) / (GAMMA_ * N_)                     #/17/


  # Load the Data Frame for this Time Step
  df_simulation_[iT_,'T']   <- iT_                        #/18/
  df_simulation_[iT_,'B_t'] <- B_t_                       #/19/

  df_simulation_[iT_,'dS']  <- dS_                        #/20/
  df_simulation_[iT_,'dE']  <- dE_                        #/21/
  df_simulation_[iT_,'dI']  <- dI_                        #/22/
  df_simulation_[iT_,'dR']  <- dR_                        #/23/
  df_simulation_[iT_,'dC']  <- dC_                        #/24/
  df_simulation_[iT_,'dD']  <- dD_                        #/25/

  df_simulation_[iT_,'S']   <- S_                         #/26/
  df_simulation_[iT_,'E']   <- E_                         #/27/
  df_simulation_[iT_,'I']   <- I_                         #/28/
  df_simulation_[iT_,'R']   <- R_                         #/29/
  df_simulation_[iT_,'C']   <- C_                         #/30/
  df_simulation_[iT_,'D']   <- D_                         #/31/

  df_simulation_[iT_,'R_e'] <- R_e_                       #/32/


}


cases_g_ <- df_csv_[,'cases_g_']
deaths_g_ <- df_csv_[,'deaths_g_']
which_is_not_na_ <- which(!is.na(cases_g_))
x_ <- df_simulation_[,'T']
y1_ <- df_simulation_[,'C']
y2_ <- df_simulation_[,'D']
main_ <- paste(main='Evolavirus 2014 - Guinea',format(nSimulations_,scientific=FALSE,big.mark=
","),'Iterations', sep=' ')
plot(x_, y1_, ylim=c(0,1000),type='l', xlab='Day Count from 31-DEC-2013',ylab='Cumulative Value
s',col='red',main=main_)
points(x_, cases_g_, type="p", pch=20, col='red')
lines(x_,y2_,type='l')
points(x_,deaths_g_, type="p", pch=20,col='black')
legend(1, 1000, legend=c("Cases", "Deaths"),col=c("red", "black"), lty=1:1, cex=0.8)
legend(1,  800,legend=c("Line: SEIR Simulation","Dots: WHO Data"), lty=1:2)
c_ <- paste("Control Interventions applied @ Day",TAU_,sep=' ')
legend(55, 1000,legend=c(c_), bty='n')
```



Now extract the Rates and Coupling Constants from the Simulation Data Frame and compare and contrast with Althaus's findings. Althaus published the following values:

- Transmission Rate β_0: 0.27
- Case Fatality Rate f: 0.72 to 0.75
- Average Frequency of Incubation σ: 0.189
- Average Frequency of Infectiousness γ: 0.178
- Exponential Decay Constant κ: 0.0023 to 0.0024 per day
-

```
print(noquote(paste('Althaus c.f. Monte Carlo Simulation for',format(nSimulations_,scientific=FA
LSE,big.mark=","),'Iterations',sep=' ')))
```

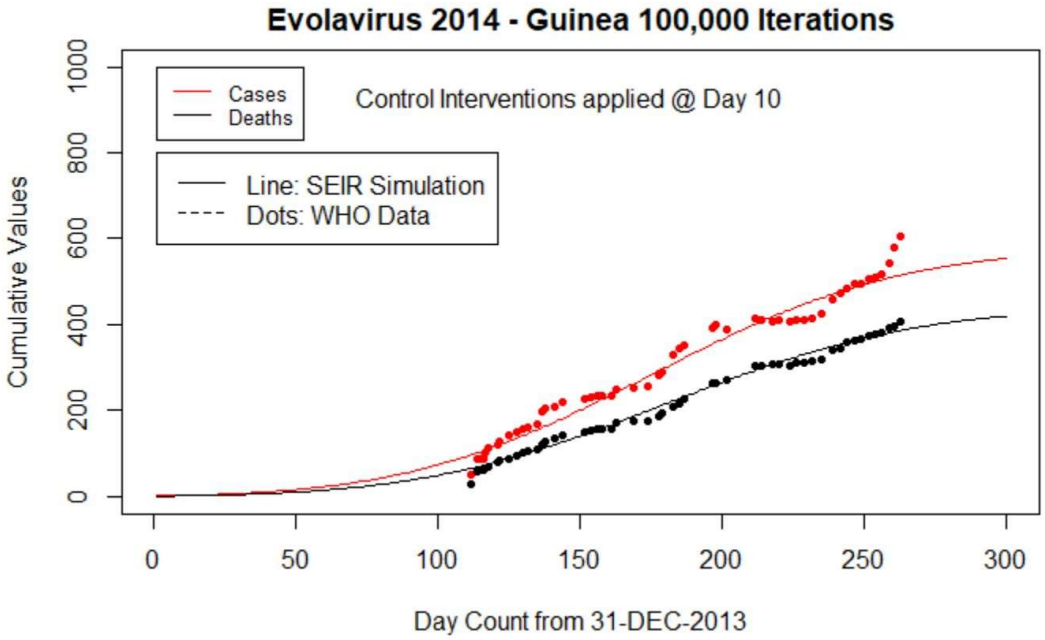
```
## [1] Althaus c.f. Monte Carlo Simulation for 100 Iterations
```

```
columns_ <- c('B0','SIGMA','GAMMA','KAPPA','F')
Althaus = df_althaus_[1,columns_]
Simulation = df_monte_carlo_[which_,columns_]
DeltaPercent = (Simulation - Althaus)/Althaus
rbind_ <- rbind('Althaus'=Althaus, '4-Vector'=Simulation, 'Delta%'=DeltaPercent)
t(rbind_)
```

```
##         Althaus    4-Vector      Delta%
## B0    0.2700000 0.266203682 -0.01406044
## SIGMA 0.1886792 0.169521496 -0.10153607
## GAMMA 0.1782531 0.173435693 -0.02702576
## KAPPA 0.0023000 0.002613034  0.13610160
## F     0.7400000 0.759681060  0.02659603
```

So with even as few as 100 iterations, we are within 15% of Althaus's published values, and we can do much better with more spend and compute.

We performed a simulation with 100,000 iterations, and obtained nearly perfect agreeement with Althaus:




```
[1] Althaus cf Monte Carlo Simulation for 100,000 Iterations
        Althaus   4-Vector      Delta%
B0    0.2700000 0.28239080  0.04589186
SIGMA 0.1886792 0.15905753 -0.15699509
GAMMA 0.1782531 0.18598076  0.04335205
KAPPA 0.0023000 0.00249238  0.08364334
F     0.7400000 0.76255110  0.03047446
```

You will notice that we appear to have a 15% difference for σ, but this is not the first time. In our first paper, we differed from Althaus also. Althaus publishes $\sigma = \frac{1}{5.3}$ but could not fit our curves in the first paper with that value. At that time, we did a small guess-and-check and found $\sigma = \frac{1}{6.5}$, which is 0.0154 and close to the 0.0159 value that we have found in simulation here. We did not call attention to this discrepancy at that time since guess-and-check is hardly a rigorous simulation, but that value has emerged again, so we can note it here as an honest difference with Althaus. Indeed, we do not know how to obtain a good SEIR curve fit to the WHO data with $\sigma = \frac{1}{5.3}$

Conclusions and Afterword on COVID-19

The machinery in this primer will give the interested data scientist all of the mathematical tools required to derive SEIR Rates and Coupling Constants for any contagion as the epidemic unfolds and epidemiological data emerges. The SEIR model is a so-called Compartmental Model, where health statuses are coupled by constants and rates that govern migration from one compartment to the next.

Note that SEIR assumes that the Removed subjects who survive the contagion gain Immunity. In the case of COVID-19, as of this writing, it seems that it has not been scientifically established that survivors develop Immunity; if that finding is firmly established, then the SEIR model will be inappropriate, and indeed the inability of the above procedure to converge to COVID-19 epidemiological data may provide a first hint that Immunity is not achieved with that contagion. In that case, one of SEIR's sibling models, such as the Susceptible-Infected-Susceptible (SIS) model would be required, as it is for the common cold or influenza. The Wiki offers an excellent discussion.

https://en.wikipedia.org/wiki/Compartmental_models_in_epidemiology (https://en.wikipedia.org/wiki/Compartmental_models_in_epidemiology)